IE 310 Fall 2024 Assignment

Beyza Nur Deniz

2021400285

Contents

1. Generation of Instances

1.1. Overview of the Method

`generate_transportation_feasible_instance` function is utilized to create a feasible instance for the transportation problem. This method takes the number of supply nodes, the number of demand nodes, maximum possible cost, and the maximum value of demand and supply nodes as parameters and returns 3 lists: supply, demand, and cost. `supply[i]` represents the i^{th} supply node's value, `demand[i]` represents the i^{th} demand node's value and `cost[i][j]` represents the cost of transportation from i^{th} supply node to j^{th} demand node. A generated instance with 3 supply nodes, 4 demand nodes, 8 maximum possible cost, and 14 maximum possible demand/supply value is given in Figure 1.

```
Supply array:  [1, 9, 13]
Demand array:  [3, 12, 7, 1]
Cost matrix:  [[5, 3, 1, 3], [6, 8, 4, 8], [7, 2, 1, 0]]
```

Figure 1: Sample Feasible Instance Generated

1.2. Meeting The Constraints

The integrality, nonnegativity, and given possible maximum value constraints are guaranteed with the usage of `random.randint()`.

Total supply and total demand matching is guaranteed by the loops after randomly created arrays. The difference between supply and demand is detected, and the function removes values from random nodes from the larger one.

2. Utilizing Solvers

Python's *PuLP* library is utilized for this step. There is not much to explain at this step since it only implements a method with the instance generated at the first step. By our generation, it is guaranteed that an optimal solution exists, so this method returns only the optimal solution and the optimal value. It is not necessary for us to use the solution status, so it is ignored.

3. Coding Revised Simplex

A general revised simplex solver utilized at the file *revised_simplex.py*, and it is used to solve the transportation problem at the file *revised_simplex_for_tranportation.py*.

3.1. General Revised Simplex Solver

The `revised_simplex(A, b, c)` method takes A matrix, b and c vectors of the LP problem, assumes they are given in the standard form, all necessary changes made, and all necessary variables added. At the revised simplex method, all iterations only need the basic variables, non-basic variables and A, b, c parts corresponding to them.

This method utilizes `find_entering_var_index(c_b, B, N, c_n)` and `find_leaving_var_index(A_i, B, b)` methods to find entering and leaving variables for each iteration until optimized. Since the transportation problem is a minimization problem, `find_entering_var_index(c_b, B, N, c_n)` is implemented accordingly.

At a maximization problem, for entering variable we took the most negative reduced cost, and concluded we are at the optimal if no negative coefficient remained. For the minimization problem, this method uses the opposite of that test; it takes the most positive one as the entering variable, and concludes we are at the optimum if no positive objective coefficient remains.

Minimum ratio test is applied to find the leaving variable. No changes are made at the minimum ratio test for the minimization problem. It takes the RHS and the $B^{-1} \cdot A_i$ and takes the minimum ratio as the leaving variable. For the minimum ratio test, non-positive values are ignored, and if we can't find a leaving variable when not at the optimum, it means the problem is unbounded.

Iterations continue until no entering variable is found, or an unbounded conclusion comes from the minimum ratio test.

3.2. Transportation Problem Adapted To Revised Simplex

To use the revised simplex implemented at the previous step, some adaptations were necessary to the feasible instance generated. Two main issues arose:

- Turning supply, demand, cost into proper A, b, c
- Because of the equality constraints how to handle initial basis

3.2.1 Transfering the problem into standard LP form

The `prepare_transportation_for_simplex(supply, demand, cost)` method is implemented for this issue. Since a 2-D A matrix is required, some calculations were necessary at the cost matrix.

The method initializes the constraint matrix A, the right-hand side vector b, and the cost vector c. The A matrix is constructed to ensure that each supply and demand constraint is represented correctly. For each supply node, constraints are added to ensure the total outgoing flow matches the supply. Similarly, for each demand node, constraints are added to ensure the total incoming flow satisfies the demand. The c vector is populated with the corresponding costs from the input cost matrix.

3.2.2 Initial Basis

The **Big-M** method is used to solve the initial basis and artificial variable problems. For every artificial variable added to the problem and constraints, the variable is added to the objective function with the coefficient M. Since the problem is a minimization problem, it is added to the objective function as $+M$. M is chosen as 10^6 since the largest possible cost was chosen as 1000 at the *main.py*. It could also be taken as a parameter, and decided there, but it was fixed at the *main.py*, so it felt unnecessary.

4. Experiments

4.1. Experimental Setup

To evaluate the performance of the implemented Revised Simplex Algorithm, multiple instances of the transportation problem were created by varying the problem size in terms of the number of supply and demand nodes. The following constraints and conditions were maintained across all instances:

- The maximum cost and maximum supply/demand quantities were fixed at 1000, ensuring consistency across all tests.
- An equal number of supply and demand nodes was maintained for each instance to simplify comparisons.

- The *Numpy* random seed was fixed to guarantee reproducibility of the experiments.

Nine problem sizes were defined to test and compare the runtime of solvers. Supply and demand sizes were capped at 250, as larger sizes required computation times exceeding 2-3 minutes, which was deemed unnecessary for practical comparisons. For each test, two random integers, `num_supply` and `num_demand`, were generated, and corresponding problem instances were created. The Revised Simplex Algorithm and the *PuLP* solver were both run on these instances, and their runtimes were recorded for comparison. Finally, a plot was generated to visualize the runtime differences as a function of problem size.

4.2. Methodology

4.2.1 Problem Instance Generation

For each of the nine problem sizes, random instances of the transportation problem were generated. The supply and demand quantities were drawn randomly while ensuring the total supply equaled the total demand to maintain feasibility. The cost matrix was also populated with random values between 0 and 1000.

4.2.2 Solvers and Timing

Both the implemented Revised Simplex Algorithm and the *PuLP* solver were run on each instance. Execution times were measured to evaluate their performance.

4.2.3 Optimality Check

The solutions obtained from both solvers were compared to ensure they matched. Additionally, the integrality of the solutions was verified to check whether all solutions were integers or if fractional solutions occurred.

4.2.4 Analysis of Growth

The runtime growth of the implemented Revised Simplex Algorithm was analyzed and compared to the *PuLP* solver across various problem sizes. The comparison is visualized in Figure 2, where the x-axis represents the case size (number of supply and demand nodes), and the y-axis represents the computation time in seconds.

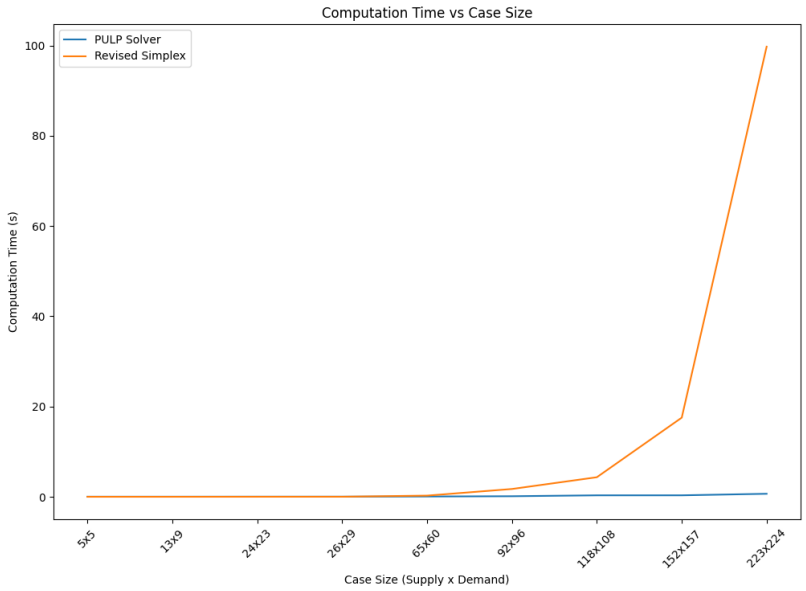

Figure 2: Runtimes Compared Accorindg To Sample Sizes

From the figure, the following observations can be made:

- For small problem sizes (e.g., 5×5 to 60×60), the Revised Simplex Algorithm performs similarly to the *PuLP* solver, with little differences in computation time. Both solvers solve these instances almost instantaneously.

- As the problem size increases beyond 65×60, the computation time for the Revised Simplex Algorithm begins to grow significantly, similar to an exponential trend. For example, at the largest problem size tested (223×224), the runtime of the Revised Simplex Algorithm exceeds 100 seconds, whereas *PuLP* was still under 1 second.

- This exponential growth is can be caused by the costliness of the matrix operations as the matrices gets bigger. Solvers like *PuLP* utilizes optimizers and other helpers to remain able to solve the problems instantaneously, however implementing revised simplex by hand, we used nothing to improve the algorithms for larger sizes.

In conclusion, the Revised Simplex Algorithm is suitable for small to moderately sized problems but becomes impractical for larger problem instances due to its exponential runtime growth. The *PuLP* solver, on the other hand, demonstrates excellent scalability and efficiency, making it the preferred choice for solving large-scale transportation problems.

```
--- Case 1 ---
Problem Size: 5x5
PULP Solver Results: Time = 0.0162s, Cost = 858835.0
Transportation Plan:
[322.0, 355.0, 266.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 55.0, 340.0]
[0.0, 0.0, 0.0, 785.0, 0.0]
[0.0, 0.0, 97.0, 0.0, 296.0]
[0.0, 324.0, 0.0, 0.0, 0.0]
Revised Simplex Results: Time = 0.0004s, Cost = 858835.0
Transportation Plan:
[322. 355. 266.   0.   0.]
[  0.   0.   0.  55. 340.]
[  0.   0.   0. 785.   0.]
[  0.   0.  97.   0. 296.]
[  0. 324.   0.   0.   0.]

--- Case 2 ---
Problem Size: 13x9
PULP Solver Results: Time = 0.0155s, Cost = 735492.0
Transportation Plan:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 76.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 192.0, 155.0, 0.0, 0.0]
[0.0, 0.0, 48.0, 0.0, 0.0, 295.0, 0.0, 0.0, 0.0]
[0.0, 200.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 0.0]
[0.0, 0.0, 0.0, 313.0, 0.0, 0.0, 125.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 97.0, 0.0, 0.0, 0.0, 694.0, 0.0]
[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 670.0, 0.0, 4.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 128.0]
[0.0, 0.0, 0.0, 0.0, 329.0, 0.0, 0.0, 0.0, 0.0]
[310.0, 142.0, 0.0, 356.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Revised Simplex Results: Time = 0.0010s, Cost = 735492.0
Transportation Plan:
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
[ 0.  0.  0.  0.  0.  0. 76.  0.  0.]
[  0.   0.   0.   0.   0. 192. 155.   0.   0.]
[  0.   0.  48.   0.   0. 295.   0.   0.   0.]
[  0. 200.   0.   0.   0.   0.   0.   0.  12.]
[ 0.  0.  0.  0.  0.  0.  0. 47.  0.]
[  0.   0.   0. 313.   0.   0. 125.   0.   0.]
[  0.   0.   0.  97.   0.   0.   0. 694.   0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[  0.   0.   0.   0. 670.   0.   4.   0.   0.]
[  0.   0.   0.   0.   0.   0.   0.   0. 128.]
[  0.   0.   0.   0. 329.   0.   0.   0.   0.]
[310. 142.   0. 356.   0.   0.   0.   0.   0.]
```

Figure 3: Terminal Output

```
--- Case 3 ---
Problem Size: 24x23
PULP Solver Results: Time = 0.0206s, Cost = 1250634.0
Revised Simplex Results: Time = 0.0074s, Cost = 1250634.0

--- Case 4 ---
Problem Size: 26x29
PULP Solver Results: Time = 0.0234s, Cost = 1152875.0
Revised Simplex Results: Time = 0.0109s, Cost = 1152875.0

--- Case 5 ---
Problem Size: 65x60
PULP Solver Results: Time = 0.0627s, Cost = 1211237.0
Revised Simplex Results: Time = 0.2529s, Cost = 1211237.0

--- Case 6 ---
Problem Size: 92x96
PULP Solver Results: Time = 0.1205s, Cost = 1101850.0
Revised Simplex Results: Time = 1.7244s, Cost = 1101850.0

--- Case 7 ---
Problem Size: 118x108
PULP Solver Results: Time = 0.3176s, Cost = 1046944.0
Revised Simplex Results: Time = 4.3302s, Cost = 1046944.0

--- Case 8 ---
Problem Size: 152x157
PULP Solver Results: Time = 0.3205s, Cost = 1241523.0
Revised Simplex Results: Time = 17.5330s, Cost = 1241523.0

--- Case 9 ---
Problem Size: 223x224
PULP Solver Results: Time = 0.6634s, Cost = 1006357.0
Revised Simplex Results: Time = 99.7541s, Cost = 1006357.0
```

Figure 4: Terminal Output

4.2.5 Integrality Testing

For integrality testing 5 more small test cases were created to observe. All solution matrices were printed to terminal. As can be seen from the outputs all solutions are integers.

```
--- Case 1 ---
Cost = 62.0
Transportation Plan:
[0. 0. 0. 0. 1.]
[0. 8. 0. 0. 1.]
[1. 0. 0. 0. 0.]
[0. 0. 0. 0. 1.]
[0. 0. 0. 0. 1.]
[0. 0. 5. 0. 0.]
[0. 0. 0. 2. 0.]
[5. 0. 0. 0. 0.]
[2. 0. 0. 4. 0.]
Problem Size: 9x6

--- Case 2 ---
Cost = 99.0
Transportation Plan:
[0. 4. 0. 0. 0. 3.]
[0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 2. 0.]
[0. 0. 5. 0. 4. 0.]
[0. 0. 0. 0. 1. 0.]
[4. 3. 0. 0. 1. 0.]
[1. 0. 0. 1. 0. 0.]
[0. 0. 0. 3. 0. 0.]
[0. 0. 0. 0. 0. 5.]
Problem Size: 5x3

--- Case 3 ---
Cost = 80.0
Transportation Plan:
[0. 1. 0.]
[0. 0. 5.]
[5. 0. 0.]
[0. 5. 2.]
[1. 0. 0.]
Problem Size: 9x9
```

Figure 5: Terminal Output

```
--- Case 4 ---
Cost = 32.0
Transportation Plan:
[0. 6. 0. 0. 1. 0. 2. 0. 0.]
[0. 0. 0. 6. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 4. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 7.]
[0. 0. 0. 0. 0. 9. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[3. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 7. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
Problem Size: 6x7

--- Case 5 ---
Cost = 59.0
Transportation Plan:
[1. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 1. 0. 2.]
[5. 0. 0. 1. 0. 1. 0.]
[0. 0. 0. 2. 0. 0. 0.]
[3. 0. 0. 0. 0. 0. 0.]
[0. 0. 2. 0. 2. 0. 0.]
```

Figure 6: Terminal Output